Many libraries re-launch Ruby to perform additional tasks. Sometimes this is done for process isolation, sometimes it's to change startup command-line for a subset of work, and sometimes it's just the easiest way to launch a clean Ruby environment. In almost every case, the command line is built using "ruby" explicitly as the command to be executed. This only works if the Ruby version/implementation you want to run is installed as "ruby" and exists on PATH. If, for example, you are running JRuby, you most likely want the subprocess to launch a new JRuby instance. I propose that we add a standard way to launch a new Ruby instance to Kernel.

Kernel#ruby would launch, in an implementation-specific way, a new instance of the current Ruby. On normal C Ruby, this would simply use the full path to the executing "ruby" binary (using appropriate name, like "/usr/bin/ruby1.9" if necessary) and launch it directly, given the specified parameters. On implementations like JRuby, which can launch many instances in the same JVM, the Kernel#ruby method could simple launch a new instance of JRuby. In both cases, it would avoid problems with expecting libraries to build their own command line (and usually getting it wrong).

The method would take, at minimum, a set of command-line parameters to the new instance. These could be in the form of an array of strings, a set of string parameters, or a single string parameter (I don't know which is best). There may be a need for non-command-line options to the #ruby method, so the array or single string might be best (for example, an option to specify whether to attempt to keep the launch in-process, using some form of MVM)

The method would launch Ruby with the given command-line parameters in some implementation-specific way.

The method could either return streams and pid, like p/open/3/4, or simply return the subprocess's output. There could also be a separate #ruby_open to do popen-like stream-driving.

The method should be backported to 1.8.7, ideally, so that libraries could freely start to use it as soon as possible.

Example usages:

```
ruby "-e \"puts 1\""
in, out, err = ruby "-EUTF-8", mvm:true
```

This won't happen (at least under current policy). Newer libraries should just drop 1.8.7 support.

Hi,

I am not against adding a method to spawn the interpreter in platform dependent way so that script can fork the interpreter without worrying
implementation detail. But I object to name it 'ruby' nor adding it to 1.8.

In message "Re: [ruby-core:34355] [Ruby 1.9-Feature#4265][Open] Provide a core method Kernel#ruby for invoking a new Ruby instance" on Wed, 12 Jan 2011 00:54:53 +0900, Charles Nutter redmine@ruby-lang.org writes:

Many libraries re-launch Ruby to perform additional tasks. Sometimes this is done for process isolation, sometimes it's to change startup command-line for a subset of work, and sometimes it's just the easiest way to launch a clean Ruby environment. In almost every case, the command line is built using "ruby" explicitly as the command to be executed. This only works if the Ruby version/implementation you want to run is installed as "ruby" and exists on PATH. If, for example, you are running JRuby, you most likely want the subprocess to launch a new JRuby instance. I propose that we add a standard way to launch a new Ruby instance to Kernel.

Kernel#ruby would launch, in an implementation-specific way, a new instance of the current Ruby. On normal C Ruby, this would simply use the full path to the executing "ruby" binary (using appropriate name, like "/usr/bin/ruby1.9" if necessary) and launch it directly, given the specified parameters. On implementations like JRuby, which can launch many instances in the same JVM, the Kernel#ruby method could simple launch a new instance of JRuby. In both cases, it would avoid problems with expecting libraries to build their own command line (and usually getting it wrong).

The method would take, at minimum, a set of command-line parameters to the new instance. These could be in the form of an array of strings, a set of string parameters, or a single string parameter (I don't know which is best). There may be a need for non-command-line options to the #ruby method, so the array or single string might be best (for example, an option to specify whether to attempt to keep the launch in-process, using some form of MVM)

The method would be backported to 1.8.7, ideally, so that libraries could freely start to use it as soon as possible.

Example usages:
[ruby "-e 'puts 1'"
|in, out, err = ruby "-EUTF-8", mvm:true

#4 - 01/12/2011 11:12 PM - now (Nikolai Weibull)

On Tue, Jan 11, 2011 at 16:54, Charles Nutter redmine@ruby-lang.org wrote:

Feature #4265: Provide a core method Kernel#ruby for invoking a new Ruby instance

Why not put it under the Process namespace instead?

#5 - 01/13/2011 03:36 PM - headius (Charles Nutter)

Process namespace would be fine as well. The nice thing about having it in Kernel is that it would be a top-level function everyone could use without dereferencing anything. And if it were named Ruby, it's a pretty clean command-line-like piece of code:
ruby '-rubygems', 'myscript.rb'

But Matz doesn't like it called 'ruby', so I'm thinking of some new ideas. Brainstorming:
(These all might demand a namespace, since some are pretty generic)
"launch"
"run"
"run_ruby" ("exec_ruby", "launch_ruby", "start_ruby", "ruby_run", etc)
"rb" ("start_rb", etc)

#6 - 01/13/2011 03:37 PM - headius (Charles Nutter)

"spawn_ruby", "popen_ruby"

#7 - 01/14/2011 04:29 AM - hongli (Hongli Lai)
Do you really need a core method for this? People can just copy-paste the code we use in Phusion Passenger for locating the current Ruby. We even have RVM support.

https://github.com/FooBarWidget/passenger/blob/6a5d5b2e502823208aad87b776cc0eaf032fe02e/lib/phusion_passenger/platform_info/ruby.rb#L45-88

It appears you'd need to include almost that entire module into a project to get all the necessary helper methods to make it work, not just the lines you linked. Over 300 lines is quite a lot for "just copy-paste", and the fact that it has to handle a wide variety of platforms and implementations makes it an excellent candidate for being supported in the standard library, doesn't it? Also, while I won't pretend to know the linked lines inside and out, I do see a few hardcoded forward slashes in paths, which don't seem too friendly to Windows users. Is there a separate method that is used for Windows, apart from the linked one?

I don't think it is a clean environment (I think it forks the parent process so you have a copy of everything it had) but does the following handle many of these cases? It gives you an isolated environment plus a way to talk to the parent process if you want.

IO.popen("-") {puts "I'm in a new ruby interpreter"}

Hongli/Michael: The linked code would probably work well enough as a default implementation. The problem is it's not default, so everyone has their own code for doing it. This could could probably be included, verbatim, as a standard Ruby library or used to implement a core class Ruby method. This bug isn't as much about how it would be implemented (it would be easy) as getting it built into Ruby core in the first place so everyone's using the same mechanism.

The point about forward vs back slashes is well made, but probably pretty easy to remedy.

Eric: That largely does work, but as you point out it doesn't give a clean environment. It also doesn't allow passing new/different command line arguments (like changing global encodings) nor loading different versions of libraries. And of course, it doesn't do anything at all on implementations that don't support fork :)

My hope was that this could be used with JRuby/1.9 MVM support to also hide whether a process were launched at all. The idiom of "re-launch the current Ruby with these arguments" is so common, it would be good to have in core so everyone launches the right instance and
implementations that don't need to spin up a new process can do so any way they choose. JRuby currently fakes this by looking for system() command lines that appear to be launching 'ruby' or 'jruby' and simply start a new JRuby instance in the same process, but that often fails (and it would be nice if the user were able to choose how it launches).

As Hongli pointed out, and as Gem.ruby shows, it's easy enough to stitch together the right bits to re-launch the current Ruby. It's not so easy to hide the fact that you're physically launching a new process...

- Charlie

#12 - 02/18/2011 12:50 AM - headius (Charles Nutter)

On Wed, Feb 16, 2011 at 5:40 PM, Roger Pack rogerdpack2@gmail.com wrote:

As Hongli pointed out, and as Gem.ruby shows, it's easy enough to stitch together the right bits to re-launch the current Ruby. It's not so easy to hide the fact that you're physically launching a new process...

Most people expect to be launching new processes from system...so if I understand you correctly, this would be a shortcut for "don't launch a new process if you don't have to" then, hence it being better to be a whole new method (IO.ruby_system perhaps ?) ?

Yeah, I agree with that. It could almost be IO.mvm_ruby where what you get back is not a PID but an object suitable for messaging with and controlling the child Ruby. Whether it runs in a separate process or not is an implementation detail that the API might hide or expose through an option. But basically, it would be a way to say "I just want to run this Ruby code in a clean environment...do that however you will".

- Charlie

#13 - 03/25/2012 02:55 PM - mame (Yusuke Endoh)

- Description updated
- Status changed from Open to Assigned
- Assignee set to matz (Yukihiro Matsumoto)

#14 - 04/10/2012 06:27 PM - matz (Yukihiro Matsumoto)

- Status changed from Assigned to Rejected

No good name candidate. Maybe done via MVM change.
Matz.